EE 341 – Lab 2

Introduction to Image Processing

Lab Objectives

The purpose of this lab is to introduce you to some basic concepts in image processing. You will learn how to read and display images in the Python + SciPy environment. You will perform simple edge detection on an image we give you as well as on an image of your own. Then, you will scale-down the provided image to create its thumbnail version and scale-up the image using bilinear interpolation. Finally, you will perform some other transformations on the image and observe the results. If you enjoy this lab, make sure to take the following higher-level undergraduate classes related to image processing that are offered by the ECE and CSE departments: EE 440, CSE 455, and CSE 457.

Task 1: Displaying images

Background

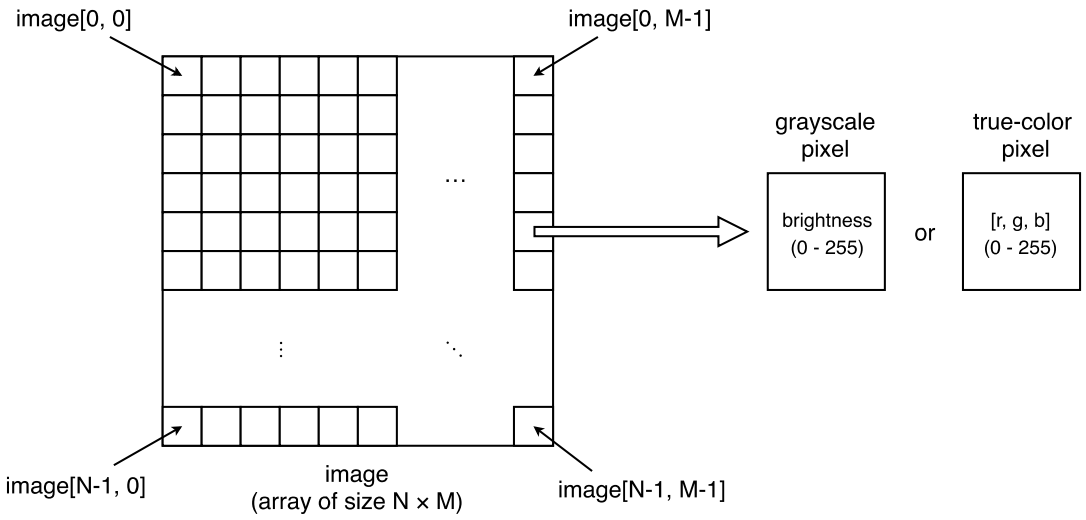


Figure 1: Array representation of image: each tiny square represents a pixel.

Digital images consist of pixels (picture elements). When the pixels are placed close to each other, images viewed on a computer display or printed on a paper appear to be continuous. The density of pixels, which is measured in the number of pixels per inch (ppi), describes the resolution of digital images. Some monitors can only display 72 ppi. For publishing, 200 – 1200 ppi is often required. Laser printers are usually capable of producing 300 – 600 ppi. The brightness and color information of each pixel is represented by a number in a multi-dimensional array. Each location in the array corresponds to the location of a pixel in the image. For example, `image[0,0]` (usually) identifies the pixel located in the upper left corner, as shown in Figure 1. Grayscale images are stored in 2D arrays, where each element in the array can take any value from 0 (black) to 255 (white). True color images are stored in 3D arrays, where the third dimension identifies the color channel. The value of the first channel: `image[y,x,0]` represents the intensity of the red component; the value of the

second channel: `image[y,x,1]` represents the intensity of the green component and the value of the third channel: `image[y,x,2]` represents the intensity of the blue component. The value of each channel ranges from 0 (darkest) to 255 (brightest). The number of bits used to encode each pixel is referred to as bit-depth or bits per pixel (bpp). The grayscale format mentioned earlier has 8 bpp while the true color format has 24 bpp (8 per each color channel).

Task 1: Displaying Image

Create a new Jupyter Notebook for this lab. Download the file `DailyShow.jpg` from Canvas and store it to the same directory where your Jupyter Notebook is stored (that is the ee341lab folder in your home directory if you followed our convention in Lab 1). Import the `numpy` and `pyplot` packages like you did in Lab 1. The image can be loaded and displayed as follows:

```
# Load the image.
image = plt.imread('DailyShow.jpg')

# Display the image.
plt.figure()
plt.imshow(image)
plt.show()
```

Since we will be working with grayscale images in this lab, your next step is to convert your input image to an 8-bit gray scale format using the `skimage` library and the function `color.rgb2gray` as below. In your notebook, include a figure of the grayscale image and output the dimensions of the image. Note: the additional argument `cmap = 'gray'` is required to display grayscale images using `imshow`. Without it, `imshow` will plot a heatmap instead.

```
from skimage import color

# Convert the image to grayscale.
image_gray = color.rgb2gray(image)

# Display the grayscale image.
plt.figure()
plt.imshow(image_gray, cmap = 'gray')
plt.show()
```

Task 2: Edge Detection

Background

Next, you will perform edge detection on your image. Edge detection is often a first step used in more complicated image processing operations. Two-dimensional convolution, appropriate for images, can be performed using the function `ndimage.filters.convolve` provided by SciPy.

```
from scipy import ndimage
result = ndimage.filters.convolve(image, kernel)
```

Convolution can be used to implement edge detection. Create the following Sobel vertical edge detection convolution kernel. This kernel is designed to respond maximally to edges running vertically relative to the pixel grid. It is a two-dimensional matrix $h_1[n, m]$ in Python notation:

$$h_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Next create the following Sobel horizontal edge detection convolution kernel. This kernel is designed to respond maximally to edges running horizontally relative to the pixel grid. It is a two-dimensional matrix $h_2[n, m]$, in Python notation:

$$h_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Now, convolve your grayscale `DailyShow` image with the two edge detection kernels described as follows.

- Assume $M1$ is the result of convolving the grayscale `DailyShow` image with h_1 (i.e. $M1$ is the row gradient of the grayscale `DailyShow` image), and $M2$ is the result of convolving the grayscale `DailyShow` image with h_2 (i.e. $M2$ is the column gradient of the grayscale `DailyShow` image).
- Use Python to display the row gradient magnitude ($|M1|$), the column gradient magnitude ($|M2|$), and the overall gradient magnitude (i.e. $\sqrt{M1^2 + M2^2}$).

Include the figures of these edge images (i.e. $|M1|$, $|M2|$ and ($\sqrt{M1^2 + M2^2}$) in your notebook.

[Optional]

You may notice lots of dark areas in these edge images. To save the toner (or cartridge) of your printer, you can try to figure out a way to reverse the grayscale of your edge images before printing them out. That is, you do a transformation to map the original darkest area to the brightest one, and the original brightest area (e.g. edge) to the darkest one.

Use Your Own images

To better understand the horizontal and vertical masks, you will now use your own image. Find an image that you like, one of your photographs or some other .jpg file off the web, which either has a lot of horizontal or vertical edges in it (or both).

Assignment 2

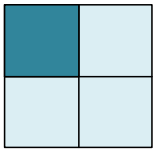
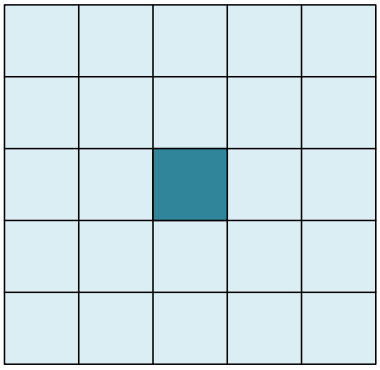
Choose an edge detector to apply to your image (horizontal if your image has horizontal edges, or vertical if your image has a lot of vertical edges, or both) to your image. Save your original image and the edge images you create. Include the original image and the edge images in your notebook. You may display the direct result of the convolution, or the edge gradient magnitudes. Either way, explicitly state your choice in comments or a text cell.

Task 3: Downscaling

Background

In this section you will investigate scaling of images in the spatial domain. You will scale the image X[n,m] in both the vertical and horizontal directions using the same scaling factor S. An example where you want to use such scaling would be creating thumbnail-sized pictures for a web page.

To perform a simple scaling – keep one out of S^2 pixels. Since this is a 2D scaling, you can keep the center pixel in each square of S^2 pixels when S is an odd number, and one of the 4 center pixels when S is even.

$S = 2$   $S = 5$

To perform a more advanced scaling operation, instead of keeping the center pixel in each square of S^2 pixels, keep the average of all of the pixels in this square.

$$\frac{\text{sum}\begin{bmatrix} & \\ & \end{bmatrix}}{4} \text{ for } S = 2; \quad \frac{\text{sum}\begin{bmatrix} \cdots & \\ \vdots \quad \ddots \quad \vdots & \\ \cdots & \end{bmatrix}}{25} \text{ for } S = 5$$

Assignment 3

Write a Python function that has an input argument for the scaling factor S. The function should read in the color `DailyShow` image, convert it to a gray scale, and then shrink the image by the scaling factor to form a thumbnail. Scale the original image with scaling factors S = 2 and S = 5.

Next, compute the average of all pixels in this square. Display all four results in your notebook. Be sure to label each image with the scale factor used to create it. For S = 2 and S = 5 respectively, compare the two scaled versions of the original picture (i.e. picking one out of S^2 pixels versus picking the average of each block with S^2 pixels). Which one is the better thumbnail image?

The average value of a submatrix can be conveniently computed using ndarray functions. The following sample extracts a 2x3 submatrix from an image and computes its average.

```
mean = image[2:4, 0:3].mean()
```

Suppose image is a 5 by 6 array with values from 1 to 30. Then image[2:4, 0:3] extracts rows 2 to 3 and columns 0 to 2, as shown in the example below.

```
>>> image
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12],
       [13, 14, 15, 16, 17, 18],
       [19, 20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29, 30]])

>>> image[2:4, 0:3]
array([[13, 14, 15],
       [19, 20, 21]])

>>> image[2:3, 0:3].mean()
17.0
```

Task 4: Flipping

Assignment 4

Guess how the following images will look compared to the original image $X[n, m]$, where $1 \leq n \leq N, 1 \leq m \leq M$:

(i) $X[N - n + 1, m]$
(ii) $X[n, M - m + 1]$
(iii) $X[N - n + 1, M - m + 1]$

Use DailyShow.jpg as the original image (i.e. X[n,m]). Verify your guesses by displaying resulting images of (i) (ii) (iii) in your notebook. You can check numpy.fliplr( ) and numpy.flipud( ) to see more information.

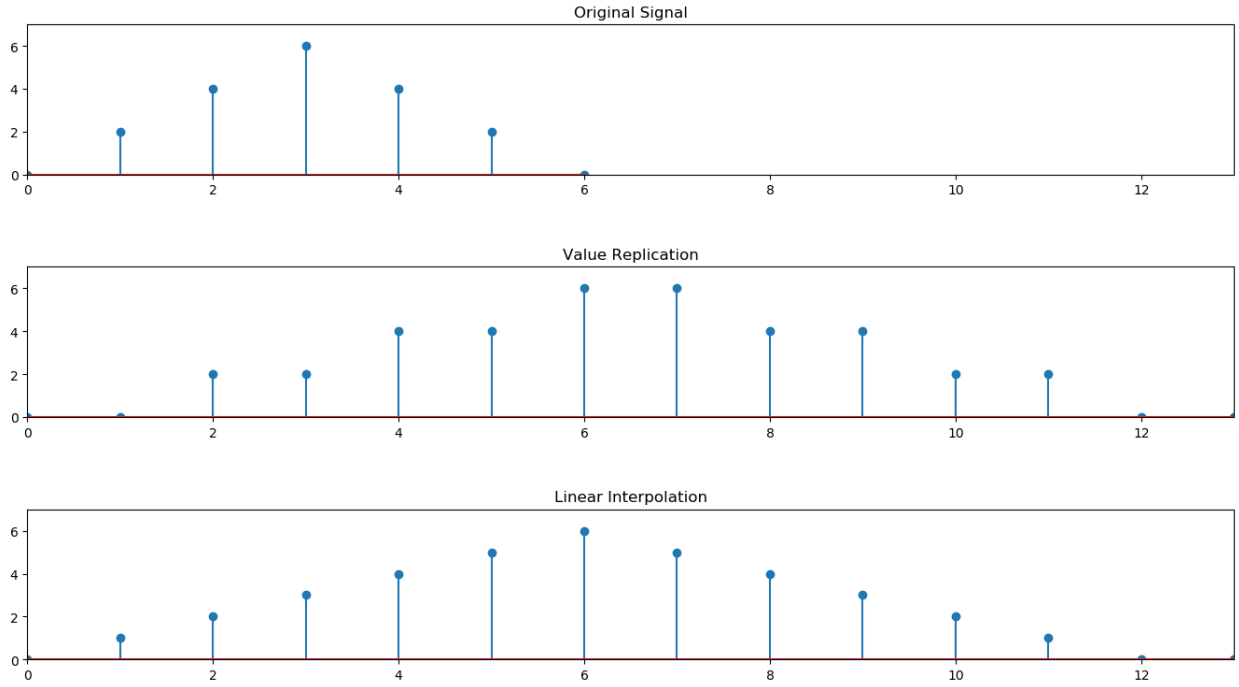
Task 5: Upscaling

Background

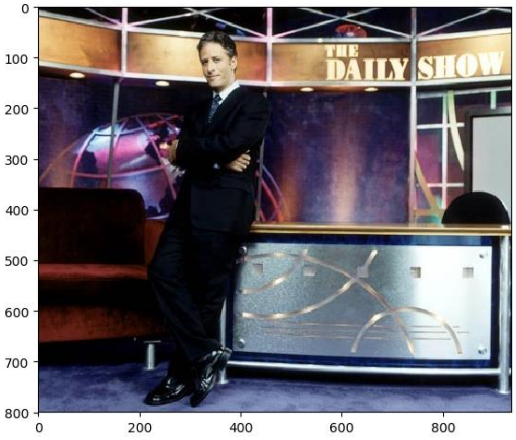
When an image is scaled up to a larger size, there is the question of what to do with the new spaces in between the original pixels. Consider a 1-dimensional signal [0, 2 ,4, 6, 4, 2, 0] and suppose we want to expand it by a factor of 2. Two possibilities for how to fill the spaces include:

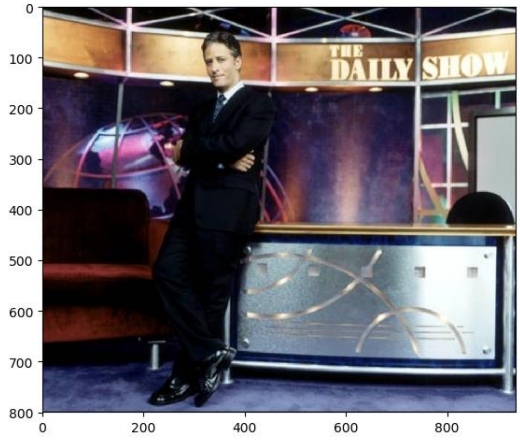
1. Simply double each sample – value replication; and
2. Make the new samples half way between the previous samples – 2 tap linear interpolation.

The figures below show the original signal and the result from each of these two methods:



The effect of the techniques on the Daily Show image are shown below.





Assignment 5

Write a Python function that can expand the input image (`DailyShow.jpg`) with dimension NxM to a 2Nx2M image using linear interpolation. Display this 2Nx2M image in your notebook. Hint: You can directly use `transform.resize()` provided by the skimage library. Figure 2 shows the concept of "bilinear interpolation" that is used to deal with a 2-dimensional signal.

```
from skimage import transform
result = transform.resize(image, (2 * rows, 2 * cols), order=1)
```

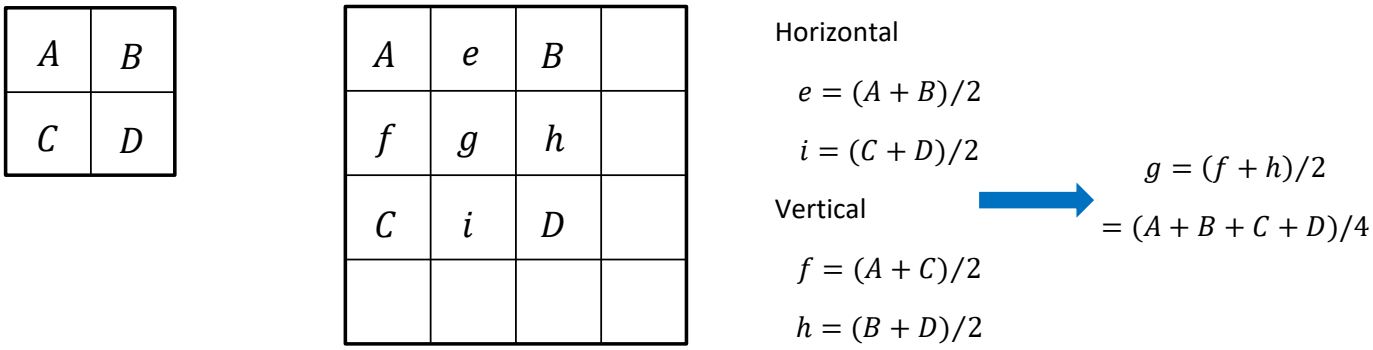


Figure 2. Enlarging the image by a factor of 2 using bilinear interpolation.

Lab Submission Requirements

Submit your completed Jupyter Notebook in `ipynb` format to Canvas. The notebook file can be obtained by clicking the menu: File – Download as – Notebook or directly accessed from the ee341lab folder in your home directory. Make sure to include the code of all the 5 tasks in code cells. In your notebook, briefly describe your results and discuss the problems you encountered and the solutions that you came up with.